Tutorial 8 : Collision

If you did the assignment from the last tutorial you probably noticed there were two noticeable flaws in it – your car could go right through the cones and would simply float if you drove off the road! In this tutorial, I'll cover the basics of how to change that.

First, here is the code with the top-down camera view implemented. I also changed the car's motion in this version from steering by rotation to a simpler side-to-side movement:

```
autocam off
sync on
sync rate 30

position camera 0,60,20
point camera 0,0,50

make object cube 1,3
position object 1,0,1.5,25

make object plain 2,12,700
position object 2,0,0,350
xrotate object 2,90

ConeZ = 50

for n=3 to 43
   make object cone n,2
   position object n,rnd(10)-5,1,rnd(3)+ConeZ
   ConeZ = ConeZ+15
next n
```

```
do
    move object 1,.5
    if leftkey()=1 then move object left 1,.2
    if rightkey()=1 then move object right 1,.2
    position camera 0,60,object position z(1)-5
    sync
loop
```

Collision detection is essentially a method of knowing when one object (in this case the "car") is overlapping another object (e.g. a cone). Once you know this has happened, you can then act appropriately. First, think about what information you need and what you want to do with that information.

In this case, I would like to know if the car is overlapping a cone (which should not happen) and I'd also like to know if the car is touching/overlapping the road which should always be the case. If the car is overlapping with a cone, I want to put it back to the last place it was before it hit the cone. If the car is no longer overlapping with the road, I want to make it fall for a given amount of time with either a "Game Over" message being displayed or simply a life being lost and the player afterwards being reset.

Here's how to do those two things.

One easy way to tell if an object is overlapping the car is to use the `object collision` command. This command takes two parameters, the id number of the first object and the id number of the object you want to check if it's overlapping. So for example, `object collision(1,2)` returns a 1 if the "car" object (with id 1) is overlapping the "road" object (with id 2) and a 0 if it isn't. I can use this in an if statement to tell the car to fall if it is no longer touching the road:

```
if object collision(1,2)=0
    for n=1 to 100
        position object 1,object position x(1),object position
        y(1)-1,object position z(1)
        center text 320,240, "Game Over"
        sync
    next n
    end
endif
```

The `object position` statements return the object's current x, y, or z coordinates, whichever you specify. Here, I tell it to simply position the object at whatever its current x coordinate is, whatever it's current y coordinate is minus 1, and whatever it's current z coordinate is. This will make the object slowly fall straight down. The center text command, centers the specified text on the given x/y coordinates. Since this game runs at 640 x 480 resolution by default, (320, 240) is the middle of the screen. Add this code inside the do-loop statements right before the sync command and see what happens.

Why did the car simply drop as soon as the game began? The reason is because I start off with the car not touching the road. So instantly the if statement detects this and triggers the end game sequence. To change this, set the car's initial y-position to 1.4 instead of 1.5

Now the code should work fine since it starts off with a little bit of the bottom of the car overlapping the road. If you ever drive off the edge though, it will detect this and correctly trigger the game over scene.

Right now, the steady drop doesn't look too realistic but you don't need to worry about that yet. That involves the issue of game physics which I plan to cover in an advanced tutorial.

But how should I check if the car is colliding with any one of the 41 cones? One way would be to have a long list of `object collision` checks:

```
if object collision(1,3)=1…
if object collision(1,4)=1…
if object collision(1,5)=1…
…
if object collision(1,41)=1…
if object collision(1,42)=1…
```

This is very inefficient, though. A different way to do it that would eliminate the repetition would be to use a `for` loop that would loop through all the cone objects and see if the car was colliding with them:

```
for n=3 to 43
     if object collision(1,n)=1…
next n
```

However, even this is unnecessary. The simplest way to check for car/cone collision is to set the second object in the collision test to 0. This is an invalid object id and when this is used, the object collision command will simply return the id of whatever object is overlapping with the first object or 0 if no objects are overlapping it. So, all I need to do to check if one of the cones is colliding with the car would be to add the following condition:

```
if object collision(1,0)>2…
```

Since only cones have id numbers greater than 2 in this game, I know that if an object with an id greater than 2 is colliding with the car, it's a cone. Now the next step is to implement the reaction of the car when it hits the cone. That's your challenge. Here are a couple tips to help you do it:

- Before moving the car at the beginning of the loop, save its current z position in a decimal variable. Decimal variables are just like regular variables except you put a # sign after them. If you wrote the following code:

    ```
    var = 2.34
    print var
    wait 2000
    ```

    it would print out 2. If you changed var to var#, though, you would get the more accurate readout of about 2.34. So also, by saving its z position in a decimal variable, you'll get a more accurate value.

- Before the sync command, check to see if it's colliding with any cones. If it is, position the car back at its old z coordinate (with the standard y-coordinate of 1.4 and it's current x coordinate) using the variable you got earlier. This way, if the car ever moves into a cone, you'll get it back in front of the cone before the player ever sees the two objects overlap.

If you had trouble understanding these hints, try reading them over again. Another way, that is easier, is to just move the car backwards by the same amount you moved it forward in the loop.